

Python for Data Science

2. Control Statements



Contents

2.1 Algorithms

2.2 Branches

2.3 Loops

1

Algorithms

Algorithm

An **algorithm** is a *procedure* for solving a problem in terms of:

1. The actions to execute.
2. The order in which these actions execute.

Program control specifies the order in which statements (actions) execute in a program.

Pseudocode

You write text that describes what your program should do.

Prompt the user to enter the first integer

Input the first integer

Prompt the user to enter the second integer

Input the second integer

Add first integer and second integer, store their sum

Display the numbers and their sum

1

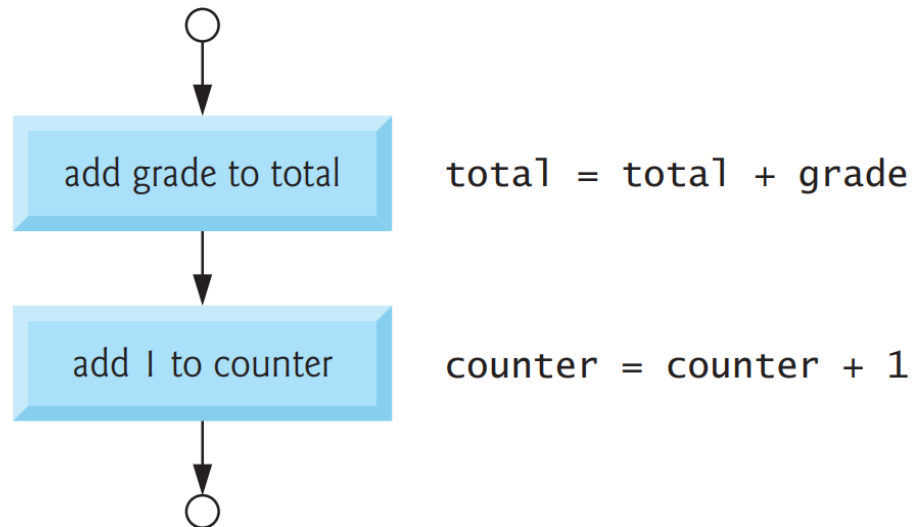
Algorithms

Forms of Control

All programs could be written using three forms of control: **sequential execution**, the **selection statement** and the **repetition statement**.

Flowcharts

A **flowchart** is a *graphical* representation of an algorithm or a part of one.



Selection Statements

Python provides three types of selection statements that execute code based on a *condition* – an expression that evaluates to either `True` or `False`.

- The `if` statement (**single-selection statement**) performs an action if a condition is `True` or skips the action if the condition is `False`.
- The `if...else` statement (**double-selection statement**) performs an action if a condition is `True` or performs a *different* action if the condition is `False`.
- The `if...elif...else` statement (**multiple-selection statement**) performs one of many different actions, depending on the truth or falsity of *several* conditions.

2

Branches

if

Suppose that a passing grade on an examination is 60.

*If student's grade is greater than or equal to 60
Display 'Passed'*

```
In [1]: grade = 85
```

```
In [2]: if grade >= 60:
...:     print('Passed')
...:
Passed
```

Indentation

Indenting a suite is required; otherwise, an `IndentationError` syntax occurs.

```
In [3]: if grade >= 60:
...: print('Passed') # statement is not indented properly
File "<ipython-input-3-f42783904220>", line 2
    print('Passed') # statement is not indented properly
    ^
IndentationError: expected an indented block
```

2

Branches

Decision

Every expression can be interpreted as either **True** or **False**.

You can base decisions on *any* expression. A nonzero value is **True**. Zero is **False**.

Strings containing characters are **True** and empty strings (' ', "" or "") are **False**.

if ... else

The **if...else** statement performs different suites, based on whether a condition is **True** or **False**.

```
In [1]: grade = 85
```

```
In [2]: if grade >= 60:
...:     print('Passed')
...: else:
...:     print('Failed')
...:
Passed
```

2

Branches

if...elif...else

You can test for many cases using `if...elif...else` statement.

The following code displays “A” for grades greater than or equal to 90, “B” for grades in the range 80–89, “C” for grades 70–79, “D” for grades 60–69 and “F” for all other grades.

```
In [17]: grade = 77
```

```
In [18]: if grade >= 90:
...:     print('A')
...: elif grade >= 80:
...:     print('B')
...: elif grade >= 70:
...:     print('C')
...: elif grade >= 60:
...:     print('D')
...: else:
...:     print('F')
...:
```


3

Loops

while

The `while` statement allows you to *repeat* one or more actions while a condition remains `True`. Such a statement often is called a **loop**.

```
In [1]: product = 3
```

```
In [2]: while product <= 50:  
...:     product = product * 3  
...:
```

```
In [3]: product  
Out[3]: 81
```

Infinite loop

Something in the `while` statement's suite must change product's value, so the condition eventually becomes `False`. Otherwise, a *logic error* called an **infinite loop** occurs.

3

Loops

for

Like the `while` statement, the `for` statement allows you to *repeat* an action or several actions.

The `for` statement performs its action(s) for each item in a **sequence** of items.

```
In [1]: for character in 'Programming':  
...:     print(character, end=' ')  
...:  
P r o g r a m m i n g
```

iterable

The sequence to the right of the `for` statement's in keyword must be an **iterable**.

```
In [3]: total = 0
```

```
In [4]: for number in [2, -3, 0, 17, 9]:  
...:     total = total + number  
...:
```

```
In [5]: total  
Out[5]: 25
```

3

Loops

break continue

The `break` and `continue` statements alter a loop's flow of control.

Executing a `break` statement in a `while` or `for` immediately exits that statement.

Executing a `continue` statement in a `while` or `for` loop skips the remainder of the loop's suite.

```
In [1]: for number in range(100):  
...:     if number == 10:  
...:         break  
...:     print(number, end=' ')  
...:  
0 1 2 3 4 5 6 7 8 9
```

```
In [2]: for number in range(10):  
...:     if number == 5:  
...:         continue  
...:     print(number, end=' ')  
...:  
0 1 2 3 4 6 7 8 9
```